VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Konzolová monitorovací utilita pro GPS čidlo

Console Monitoring Utility for GPS Sensor

2014 Michaela Bogoczová

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Michaela Bogoczová**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Konzolová monitorovací utilita pro GPS čidlo
Console Monitoring Utility for GPS Sensor

Zásady pro vypracování:

Úkolem práce je navrhnout a realizovat konzolovou aplikaci pro OS Linux využívající knihovnu Ncurses. Program se skrze standardní sériový port připojí k čidlu GPS a bude zobrazovat základní i detailní informace, které bude čidlo poskytovat.

1. Seznamte se a stručně popište protokol NMEA.
2. Popište práci s knihovnou Ncurses.
3. Navrhněte a realizujte program, který bude fungovat jako konzolová aplikace. Program bude běžet v nekonečné smyčce a bude zobrazovat aktuální informace o stavu GPS čidla, jako například počet používaných GPS satelitů, počet viditelných satelitů, sílu signálu, zeměpisnou polohu, rychlost pohybu, aktuální čas a případně další.
4. Při programování dbejte všech dobrých zásad pro tvorbu programů v OS Linux.
5. Program by měl být dobře konfigurovatelný, stabilní a jednoduchý, tak aby ho bylo možné využívat ve stálém provozu.
6. K programu přiložte dokumentaci.

Seznam doporučené odborné literatury:

[1] kolektiv autorů. Linux - Dokumentační projekt. 4. vydání. Brno: Computer Press, 2007. ISBN: 978-80-251-1525-1
[2] Geisshirt, Kenneth. Pluggable Authentication Modules: The Definitive Guide to PAM for Linux SysAdmins and C Developers. 1. vydání. Birmingham: Packt Publishing Ltd., 2007. ISBN 978-1-904811-32-9

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Seidl, Ph.D.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 7.května 2014

Bogocová .....

Mé poděkování patří panu Ing. Davidu Seidlovi, Ph.D. za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování bakalářské práce věnoval.

Abstrakt

Náplní mé práce je vytvoření konzolové aplikace pod OS Linux. Aplikace bude sloužit ke snadnému přístupu k GPS čidlu a čitelnému výstupu dat pro uživatele. Výstup bude reprezentován graficky pomocí knihovny Ncurses. GPS čidlo poskytuje data pomocí protokolu NMEA 0183, který bude zpracovávat svobodná knihovna libnmea.

Klíčová slova: GPS, NMEA, klient, Ncurses, terminál, C++, Linux

Abstract

My job is to create a console application under Linux OS. The application will be used to easily access the GPS sensor and readable data output to the user. The output will be represented graphically using ncurses. GPS sensor provides data using NMEA 0183 protocol, which will handle the free library libnmea.

Keywords: GPS, NMEA, client, Ncurses, console, C++, Linux

Seznam použitých zkratek a symbolů

GLONASS	–	Globalnaja navigacionnaja sputnikovaja sistěma
GPS	–	Global Positioning System
NAVSTAR GPS	–	Navigation Signal Timing and Ranging Global Positioning System
NMEA	–	National Marine Electronics Association
POSIX	–	Portable Operating System Interface
PPS	–	Precision Positioning Service
SPS	–	Standart Positioning Service
USB	–	Universal Serial Bus
USAF	–	United States Air Force

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Potřeba určit polohu a směr pohybu provází člověka od nepaměti. Lidé se v dávné minulosti snažili nejprve orientovat pomocí Slunce a hvězdné oblohy. Od té doby uběhlo již několik tisíc let, během kterých se navigační systémy staly každodenní nepostradatelnou součástí dnešního světa, kdy k určení polohy je nutné mít pouze funkční GPS přijímač, který se spojí se satelity na oběžné dráze a určí s jejich pomocí naše souřadnice na Zemi. GPS přijímač se dnes dá často zakoupit jako již integrovaný v automobilu, mobilním telefonu nebo tabletu. Vzhledem k rozšířenosti GPS služby existuje velké množství grafických GPS aplikací, avšak konzolové (grafické/pseudografické) tak rozšířené nejsou, mým cílem bylo tedy jednu takovou vytvořit.

Pro svou práci jsem si vybrala GPS modul Navilock NL-402U, který se připojuje k počítači přes USB rozhraní, podporuje standartní protokol NMEA 0183 a je pomocí něj možnost přijímat signál GPS i signál evropského navigačního systému Galileo. Na základě přijatých dat je přijímač schopen zobrazit např. informace o aktuální pozici, přesném čase, pozici družic a jejich signálu. Získané informace zasílá jako ASCII řetězce. Uvedené řetězce lze snadno dekódovat s použitím komunikačního protokolu NMEA. Díky takto získaným datům a s pomocí knihovny Ncurses, která mi umožnila využívat barevné rozhraní terminálu a vytvořit např. okna nebo menu jsem byla schopna takovou terminálovou aplikaci naprogramovat.

2 Teoretický rozbor

2.1 Satelitní navigační systémy

V této části se pokusím shrnout obecné informace o satelitních navigačních systémech od historie až po současnost. Při psaní této kapitoly jsem vycházela zejména z informační stránky Koordinační rady ministra dopravy pro kosmické aktivity [4] a z dalších zdrojů [2] [3], uvedených v seznamu referencí.

2.1.1 Historie a vývoj

Myšlenka, že při umístění vysílače na oběžnou dráhu Země by bylo možné pomocí Dopplerova posunu lokalizovat přijímač na povrchu Země, vznikla v laboratořích aplikované fyziky na Univerzitě Johnse Hopkinse. Do té doby existoval pouze regionální rádiový polohový systém LORAN, který přijímal signály z pozemních vysílačů.

První družicový navigační systém vznikl v roce 1964 v USA a nazýval se TRANSIT. Zahrnoval 6 družic, obíhajících po oběžné dráze ve výšce 1075 km a 3 pozorovací stanice na území USA. Původně byl uveden do provozu pro potřeby vojenského námořnictva s přesností určení polohy několika stovek metrů, poté od roku 1967 uvolněn pro civilní použití, zejména pro lodní dopravu. Přesnost určení polohy se postupně zvyšovala až na desítky metrů. Získané souřadnice byly pouze dvourozměrné, takže nebylo možné systém využívat pro leteckou dopravu. Další nevýhodou bylo to, že signál byl dostupný pouze občasně. Od roku 1996 je považován za ukončený, byl plně nahrazen systémem GPS. Podobné nevýhody měl i navigační systém CYKLON, který vznikl v bývalém Sovětském svazu koncem 60.let a dnes se již také nepoužívá.

Na globální polohový družicový systém TRANSIT navazuje nejrozšířenější družicový navigační systém, který stojí na vrcholu vývoje rádiových navigací – GPS. Původní název systému byl NAVSTAR GPS a jeho vývoj začal v roce 1973. Na jeho počátku stálo sloučení 2 systémů – TIMOTION (pro přesné určování času) a System 621B (pro určení polohy) do jednoho. Původně byla GPS využívána jen armádou, například pro sledování pozic vojenských jednotek a zaměřování cílů. Plné operační schopnosti bylo dosaženo v roce 1994, kdy byla na orbitu umístěna kompletní sestava 24 družic. V této fázi poskytoval systém 2 služby, první službu, označovanou jako SPS, mohli využívat všichni včetně neautorizovaných uživatelů, podmínkou bylo samozřejmě být vlastníkem GPS přijímače, druhá služba, PPS, byla pouze pro uživatele, vlastnící licenci od americké vlády, tudíž pro armádu. Služby se od sebe lišily v přesnosti lokalizace. Od konce 90. let minulého století dochází k modernizaci a vynášení dalších družic s navigačními signály na oběžnou dráhu. V současné době je v případě služby PPS chyba určení polohy lepší než 10 metrů v horizontální rovině, u služby SPS potom lepší než 20 metrů. Odchylka je způsobena následujícími vlivy:

- zpoždění signálu v ionosféře
- zpoždění signálu vlivem počasí
- vychýlení družice z udávané polohy

- nepřesnost satelitních hodin
- příjem falešných odražených signálů
- šum přijímače
- šum vysílače
- chyba na straně člověka

Konkurentem GPS je ruský systém GLONASS, jehož vývoj začal již v roce 1970, ale do provozu se podařil uvést až v roce 1996. Kvůli pádu Sovětského svazu, a tím pádem nedostatku financí, byl v provozu pouze krátkou dobu. Od roku 2003 začali Rusové pracovat na jeho znovuzprovoznění a v roce 2011 dosáhl systém opět plné funkčnosti a umožňuje úplné celosvětové pokrytí.

Některé dnešní navigační systémy jsou schopny přijímat signály z družic GPS i GLONASS zároveň, což je samozřejmě výhodné, protože takový přijímač je schopen přijmout signál až z dvojnásobku družic, a tím vypočítat pozici s vyšší přesností.

Obdobou systémů GPS a GLONASS je evropský projekt Galileo, jehož výstavba je hrazena z rozpočtu Evropské unie. Jeho velkou výhodou je to, že se jedná o projekt řízený a spravovaný civilní správou, na rozdíl od obou současných systémů, které jsou vojenské. Aktuálně se ve vesmíru nachází 4 družice vynesené v letech 2011 a 2012 a dne 12. března 2013 byla poprvé zaměřena pozemní lokalita. První služby by měl začít poskytovat na konci letošního roku (2014). Předpokládá se, že systém Galileo dosáhne plné funkčnosti na přelomu let 2019 a 2020, v té době by mělo být nad povrchem Země ve výšce přes 23 000 km 30 družic ve 3 oběžných drahách, přičemž každá dráha by měla mít 9 pozic pro družice a jednu pozici záložní. Tento systém by měl poskytovat větší přesnost lokalizace a větší rozsah navigačních služeb. Bude se jednat o 4 navigační služby:

- základní službu
- komerční službu
- službu pro životně kritické aplikace
- veřejnou regulovanou službu – jen pro oprávněné uživatele

2.1.2 Systém GPS

2.1.2.1 Vlastní systém GPS je pasivní systém sloužící k určení přesného času a polohy na jakémkoliv místě na Zemi a v jejím blízkém okolí za jakýchkoliv klimatických a povětrnostních podmínek. Systém GPS je složen ze 3 segmentů:

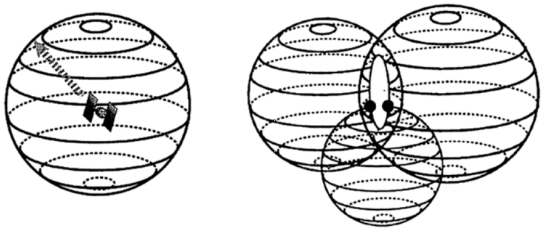
- kosmický segment
- řídící segment
- uživatelský segment

Obrázek 1: Oběžné dráhy družic nad povrchem Země. Zdroj: `http://lms.seos-project.eu/learning_modules/#17`

Kosmický segment je tvořen družicemi. Návrh původně zahrnoval 24 družic, po osmi ve třech oběžných drahách, později byl návrh upraven na vždy 4 družice v šesti oběžných drahách ve výšce přes 20 000 km nad povrchem Země, což je ilustrováno na obrázku č.1 . Dráhy inklinují 55° k rovině rovníku. Dnes se požívá 32 družic, z nichž 24 je operačních, 3 záložní ve vesmíru a 5 záložních na Zemi. Doba oběhu družic kolem Země je 11h 58min. Nejdůležitější částí družic jsou velmi přesné atomové hodiny, jejichž úkolem je zajišťovat stabilitu frekvence vysílaného signálu. Družice vysílají směrem k Zemi signály v podobě elektromagnetických vln na frekvencích označovaných jako L1 a L2. Frekvence L1 je rovna 1575,42 Mhz a je určena pro civilní uživatele, dále je tu vysílán šifrovaný kód pro autorizované uživatele, frekvence 1227,6 Mhz, nebo-li L2, je pouze pro armádu.

Řídící segment zahrnuje stanice na zemském povrchu, které permanentně monitorují kosmický segment a vyhodnocují chování jednotlivých družic. Skládá se z velitelství, nacházejícího se na letecké základně Los Angeles v Californii v USA, řídícího střediska, na Schrieverově letecké základně USAF v Colorado Springs, 3 vysílacích stanic, umístěných na základnách USAF, které mohou korigovat oběžné dráhy družic vysláním zprávy o jejich změnách, na jejichž základě si družice opraví navigační zprávu vysílanou uživatelům, nebo provádět korekci palubních hodin a 18-ti bezobslužných monitorovacích stanic umístěných na základnách USAF.

Uživatelský segment zahrnuje uživatele, kteří jsou pomocí přijímačů schopni přijímat signály z jednotlivých družic. GPS přijímače se skládají z antény, která je naladěna na frekvenci vysílanou satelity, procesoru a vysoce stabilních hodin a mohou rovněž obsahovat i displej. Přijímač je často popisován počtem kanálů, toto číslo udává počet satelitů, které je možno sledovat současně. Z bezpečnostních důvodů je systém pasivní, takže přijímače



Obrázek 2: Princip určení polohy pomocí 3 družic. Zdroj: Čábelka, M. Úvod do GPS, Praha: Přírodovědecká fakulta UK, 2008.

nemohou být zaměřeny.

2.1.2.2 Princip měření Nejdůležitější podmínkou pro příjem signálu je přímá viditelnost oblohy. Družice vysílají dálkoměrné časové značky, které přijímač z přijatého signálu detekuje a je na jejich základě schopen vypočítat polohu. Přijímač umí vypočítat vzdálenost od satelitu z rozdílu mezi odesláním a přijetím jedné sekvence kódu. Přesný čas vyslání signálu může pozorovatel zjistit díky atomovým hodinám, nacházejícím se na palubě družic. Časový rozdíl je však zatížen chybou hodin přijímače, kde se využívají méně přesné křemíkové hodiny, protože atomové hodiny jsou velmi nákladné a náročné zařízení. Na základě znalosti rychlosti šíření elektromagnetických vln, která je rovna rychlosti světla ve vakuu (cca 300 000 km/s), se dá vypočítat pozice přijímače vzhledem k vysílačům. Ke zjištění polohy přijímače, se musí ale získat signál nejméně ze 3 družic, viz. obrázek č.2 . Je-li známa poloha tří družic a jejich vzdálenost od přijímače, získají se tři kulové plochy, jejichž průnikem jsou dva body, z nichž jeden je možné vyloučit za předpokladu, že se pozorovatel nachází na zemském povrchu. Kvůli nesynchronizaci hodin přijímače a vysílače je ale potřeba ještě další družice, pro zpřesnění polohy přijímače a určení nadmořské výšky, avšak čím větší počet družic má pozorovatel v dosahu, tím je měření samozřejmě přesnější. Systém GPS je navržen tak, že z každého místa na Zemi by mělo být viditelných alespoň 6 družic, maximálně však 12.

2.2 Protokol NMEA

Komunikace přijímače s počítačem probíhá prostřednictvím protokolu NMEA 0183, který vytvořila asociace NMEA. Na svých oficiálních internetových stránkách[1] tato asociace uvádí, že dokument popisující standart NMEA 0183 je chráněn autorskými právy, pokusím se tedy alespoň popsat vlastní poznatky.

Protokol je čitelný, jedná se data v ASCII sadě. Každá zpráva začíná znakem `$` a je zakončena sekvencí `<CR><LF>`. Například zpráva obsahující čas a jiné informace vypadá jako výpis č.1, který je dekódovaný v tabulce č. 1 .

```
$GPRMC,124730.000,A,4952.2589,N,01826.2120,E,0.00,0.00,200414,,,A*6E
```

[1] https://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp

Pořadí	Formát	Příklad	Popis
1	hhmmss.sss	124730.000	Čas (UTC)
2	c	A	Status (A=OK, V=varování)
3	ddmm.mmmm	4952.2589	Zeměpisná šířka
4	c	N	Indikátor sever/jih (N=sever, S=jih)
5	ddmm.mmmm	01826.2120	Zeměpisná délka
6	c	E	Indikátor východ/západ (E=východ, W=západ)
7	d.d	0.00	Vodorovná rychlost (Speed Over Ground, v uzlech)
8	d.d	0.00	Kurz pohybu ve stupních
9	ddmmyy	200414	Datum ddmmyy
10	d.d	N.A.	Magnetická deklinace ve stupních
11	c	N.A.	Indikátor východ/západ (E=východ, W=západ)
12	xx	6E	Kontrolní součet

Tabulka 1: Zpráva nmea

Výpis 1: Příklad výstupu NMEA protokolu z GPS čidla

2.3 Knihovna ncurses

Knihovna ncurses byla vyvinuta pod operačním systémem Linux a je volně šiřitelnou emulací knihovny curses ze System V Release 4.0. Poskytuje rozhraní pro tvorbu aplikací v textovém režimu, běžících v unixovém terminálu. Je součástí projektu GNU, je však vydána pod zvláštní licencí.

Knihovna curses udržuje dvě datové struktury:

- logickou obrazovku
- fyzickou obrazovku

Fyzická obrazovka obsahuje aktuální podobu textu na obrazovce, zatímco logická obrazovka uchovává to, co by na obrazovce mělo být zobrazené. Až po zavolání funkce pro aktualizaci fyzické obrazovky, kdy knihovna porovná obsah obou obrazovek a rozhodne o optimálním postupu, jak uvést obrazovky do souladu, se obnoví fyzická obrazovka podle zjištěných rozdílů, čímž je umožněna velmi efektivní aktualizace obrazovky. Tato vlastnost vychází z kritické náročnosti na množství přenesených dat, kdy se v minulosti UNIXové stroje připojovaly po sériových linkách jako terminálová okna k serveru. Vždy existuje alespoň jedno okno, jehož velikost je shodná s obrazovkou terminálu. Poté je možné vytvářet další menší okna, která se mohou překrývat nebo mít další podokna.

Knihovna poskytuje také speciální datovou strukturu, která se nazývá plocha a slouží pro práci s informacemi, jež se do normálního okna, které nesmí velikostně přesahovat fyzickou obrazovku, nevejdou. Nejprve je nutné vytvořit celou podobu logické obrazovky

(plochy) a poté, předáním parametrů funkci pro aktualizaci plochy, je možné do různých pozic obrazovky vykreslit různé výřezy plochy.

Do všech funkcí knihovny se zadává souřadnice řádku a sloupce, což představuje konkrétní znak a jeho atributy v dané pozici obrazovky. Množství nabízených atributů je samozřejmě závislé na podpoře fyzického terminálu, většinou bývá k dispozici alespoň tučné a podtržení. Ncurses podporuje také barvy a podbarvení, přičemž barvu popředí a pozadí znaku je nutné definovat jako dvojici.

Ncurses nabízí snazší metodu pro práci s klávesnicí. Čtení z klávesnice se může řídit několika režimy, které se pomocí funkcí dají nastavovat. Znaky mohou být zpracovány po řádcích nebo ihned po zapsání, přičemž speciální znaky mohou být povoleny nebo zakázány. Knihovna používá terminfo formát. Při překladu funkčních kláves, se pro každý terminál do struktury terminfo zaznamená escape sekvence, kterou každá z jeho funkčních kláves posílá. Po začlenění hlavičkového souboru s množinou maker, v nichž jsou tyto logické klávesy definovány a zapnutí překladu funkčních kláves, převezme ncurses zpracování escape sekvencí. Výsledkem čtení logických kláves z klávesnice je tedy příslušná konstanta.

Výhodou ncurses je, že se s ní pracuje podstatně snáze než s obecným terminálovým rozhraním, protože obsluhu závislou na terminálu automaticky zpracovává knihovna a odpadá nutnost napsání velkého množství kódu na nízké úrovni. Existuje také grafická knihovna nízké úrovně svgalib, která je však obvykle k dispozici pouze pod Linuxem. Veškeré poznatky o ncurses jsem získala z knihy uvedené v seznamu použité literatury [1] .

3 Vlastní implementace

V této části se budu věnovat popisu koncepce aplikace a jejího zdrojového kódu. Výpisy kódu většinou obsahují C komentáře, ve kterých je upozorněno na odstraněné části kódu nebo na to, že by měl kód pokračovat.

3.1 Detekce GPS modulu

První věc, kterou aplikace udělá je to, že se pokusí vyhledat připojený GPS modul. V UNIXových systémech se veškerá zařízení objevují v adresáři /dev. Pro implementaci detekce předpokládám, že modul bude připojený pomocí USB a bude se tvářit jako sériový port, což znamená, že se objeví buď jako ttyUSBx nebo ttyACMx, kde x představuje číslici, kterou systém určí. Ve většině Linuxových systémech nastaví systém při připojení prvního zařízení, ttyACMx, x na hodnotu 0. Pokud se připojí druhý modul, mělo by mu být přiděleno x v hodnotě 1. Toto chování není ovšem pevně dané, zařízení může mít totiž i jiný název nebo si systém může pamatovat předchozí zařízení, které se chovaly jako sériové porty a pevně jim hodnotu x přiřazovat. Ze zkušenosti můžu říct, že se mi po častém připojování a odpojování zařízení stalo to, že se ttyACM0 začalo objevovat jako ttyACM1, dokud jsem systém nerestartovala.

```
string najdiGps(int argc, char* argv[])
{
    // Zadano jako parametr pri spousteni ?
    if ( argc == 2 )
    {
        return string(argv[1]) ;
    }

    static const char* ADRESAR_ZARIZENI = "/dev/";

    DIR *dev = opendir(ADRESAR_ZARIZENI);
    struct dirent *soubor;
    vector<string> seznam;

    // Sestavit seznam vhodnych zarizeni
    while((soubor = readdir(dev))!=NULL)
    {
        if (strncmp("ttyUSB",soubor->d_name,6) == 0 || strncmp("ttyACM",soubor->d_name,6) ==
            0)
        {
            seznam.push_back(string(soubor->d_name));
        }
    }
    closedir(dev);

    // Pokud prazdny, tak neni dostupne GPS
    if (seznam.size() == 0)
    {
        return string () ;
    }
```

```
    // z bytek kodu chybi
```

Výpis 2: Automatické vyhledání GPS modulu

Pokud nedojde k detekci zařízení, tak se aplikace nespustí. Pro všechny případy jsem tedy přidala možnost specifikovat úplnou cestu k zařízení, ať je kdekoliv. Cestu je možné specifikovat jako parametr při spouštění aplikace, v tomto případě se autodetekce samozřejmě úplně přeskočí. Detaily detekce jsou viditelné ve výpisu č.2. Pokud ovšem program najde více zařízení, které by mohly být GPS přijímače, tak vypíše jednoduché menu pro výběr pomocí std::cout.

3.2 Čtení dat z GPS modulu

Po připojení modulu NAVILOCK NL-402U do Linuxového systému se modul přidá jako zařízení ttyACM0. V UNIXových systémech se k zařízením obecně přistupuje jako k souborům, to znamená, že v mé aplikaci budu z tohoto sériového portu číst data jako z obyčejného textového souboru.

```
int mGps = open(_this->mGpsCesta.c_str(), O_RDONLY);
if (mGps == -1) {
    nmea_parser_destroy(&mNmeaParser);
    pthread_exit(0);
}
```

Výpis 3: Otevření zařízení GPS

Otevření souboru jsem provedla voláním funkce open 3, jenž mi vrátí souborový deskriptor, kterým se budu odkazovat při čtení dat. Do volání funkce open jsem jako parametr předala příznak pouze ke čtení, protože na sériový port nebudu zapisovat. Samotná data jsem získala pomocí funkce read, které jsem předala deskriptor, pole, do kterého chci byty načíst a počet znaků, které očekávám, že funkce zpracuje. Výsledkem funkce je počet přečtených bytů.

```
while((precteno = read(mGps,buffer+posun, 1)) > 0)
{
    if ( buffer[posun] == '\n' )
    {
        buffer[posun+1] = 0;
        posun = 0;
        // zpracovani radku
     }
    posun += precteno;
}
```

Výpis 4: Čtení ze zařízení GPS

Čtecí smyčka funguje na principu načítání po jednom bytu, kde se zkontroluje, zda-li poslední znak není ukončení řádku, viz. ukázka č.4 . Takto připravený řádek se dále pošle do knihovny libnmea, která text zparsuje.

Datový typ	Název	Popis
int	smask	Maska specifikující druhy balíků ze kterých byla data získána
nmeaTIME	utc	Čas UTC
int	sig	Kvalita GPS (0 = Neplatná; 1 = Fix; 2 = Diferenční, 3 = Citlivá)
int	fix	Režim operace (1 = Nedostupný; 2 = 2D; 3 = 3D)
double	PDOP	DOP Pozice
double	HDOP	DOP Horizontální
double	VDOP	DOP Vertikální
double	lat	Šírka v NDEG - +/-[stupně][min].[sec/60]
double	lon	Délka v NDEG - +/-[stupně][min].[sec/60]
double	elv	Výška nad mořem
double	speed	Pozemní rychlost v km/h
double	direction	Úhel směru pohybu
double	declination	Magnetic variation degrees (Easterly var. subtracts from true course)
nmeaSATINFO	satinfo	Informace o satelitech

Tabulka 2: Obsah struktury nmeaINFO

3.3 Libnmea

Knihovna NMEA poskytuje rozhraní pro parsování NMEA výstupu. Použití této knihovny je velice snadné, stačí pouze nainicializovat datovou strukturu pro parser, protože libnmea je naprogramovaná v C. Dále je třeba už jen zavolat parsovací funkci a předat jí parsovací strukturu, text k parsování a výstupní strukturu. Výsledky jsou ve struktuře nmeaINFO, znázorněné v tabulce č. 2

3.4 Ncurses

Knihovnu ncurses využívám pro čtení stisknutých kláves na klávesnici. Jelikož je navigace mezi obrazovkami v mojí aplikaci řešena pomocí stisku funkčních kláves F1-F4 a šipek, je třeba ncurses nastavit režim keypad pro stdscr voláním keypad(stdscr, TRUE).

Dříve než začnu pomocí ncurses kreslit nebo číst z klávesnice, musím ji inicializovat a před ukončením programu deinicializovat, v jiném případě by to mohlo způsobit problémy v terminálu. Součástí inicializace je také vytvoření barevné palety, kterého docílím pomocí jakýchkoliv konstant, v mém případě enumerátoru, propojením barevné dvojice pozadí a písma s konkrétní konstantou. Touto konstantou se poté můžu kdykoliv odkazovat na mnou připravenou paletu. Dále je také nutné terminálu nastavit, aby nevypisoval stisklé klávesy, protože by to způsobovalo tisknutí rušivých znaků do aplikace v pozici kurzoru. Tento proces detailně popisuje výpis č.5

```
void Obrazovka::pripravCurses()
```

```
{
    static bool pripraveno = false;
    if (!pripraveno)
    {
        pripraveno = true;

        initscr () ;
        curs_set(0) ;
        noecho();
        cbreak();
        keypad(stdscr, TRUE);
        if (!has_colors()) {
            endwin();
            std::cerr << "Chyba-terminal nepodporuje barvy.\n";
            return;
        }
        if ( start_color () != OK) {
            endwin();
            std::cerr << "Chyba-inicializovat barvy.\n";
            return;
        }

        init_pair (E_BARVA_POZADI, COLOR_BLACK, COLOR_WHITE);
        init_pair (E_BARVA_FUNKCE, COLOR_WHITE, COLOR_BLACK);
        init_pair (E_PODBARVENI, COLOR_RED, COLOR_WHITE );
    }
}
```

Výpis 5: Jednorázová příprava ncurses pro použití

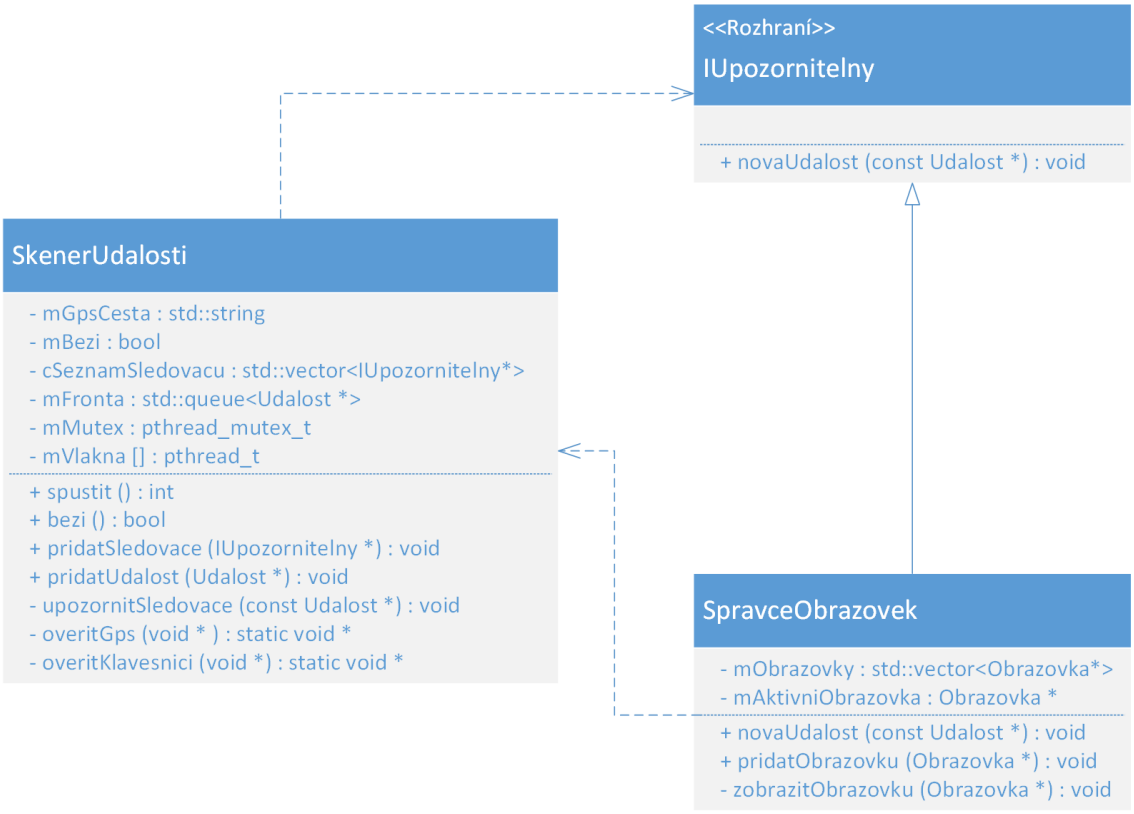
3.5 Aplikační logika

V této kapitole budou následovat třídy aplikační logiky, které mají za úkol reagovat na vstupy, distribuovat informace nebo je reprezentovat.

3.5.1 Skener událostí

Jádrem celé aplikace je jednoduchá aplikační smyčka, která kontroluje frontu událostí. Do fronty se dostávají události vzniklé ve dvou vláknech, vytvořených ve třídě SkenerUdalosti. Jedno vlákno slouží pro čtení z klávesnice a druhé pro čtení dat z GPS přijímače.

Utvoření vláken je implementováno pomocí nativní POSIX knihovny pthread. Obecně může použití vláken způsobit problémy, jelikož běží současně s hlavním vláknem aplikace, takže je třeba ošetřit, jak a kdy se budou vyměňovat data mezi hlavním a vedlejším vláknem. Tento problém obecně řeší 2 způsoby práce s vlákny, a to pomocí semaforů nebo mutexů. Já jsem použila mutexy, a tak se semaforům věnovat nebudu. Mutex funguje jako zámek pro uzamčení určité části kódu. Před prvním použitím je třeba mutex inicializovat. Dále se musí určit, jaké proměnné budou mezi hlavním a přídavným vláknem společné a všechna místa, na kterých se ke společné proměnné přistupuje, zabalit voláním pro uzamčení a odemčení mutexu. Uzamčení funguje tak, že ten kdo uzamkne zámek jako



Obrázek 3: UML třídní diagram skeneru událostí

první, pokračuje dál a pracuje s uzamčenou částí. Ve chvíli, kdy se někdo zámek pokusí uzamknout podruhé, se jeho vykonávání kódu právě zde zastaví a vyčkává na opětovné uvolnění zámku.

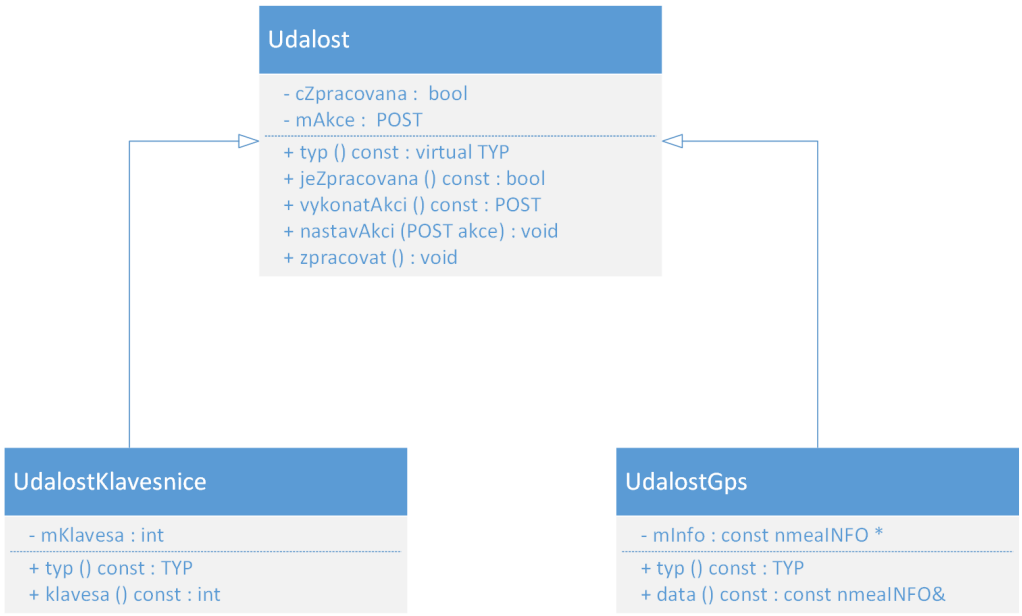
Samotný kód vlákna se předává jako ukazatel na funkci při startování vlákna. Chtěla jsem, aby kód vlákna byl ve třídě SkenerUdalosti, ale narazila jsem na problém, spočívající v tom, že nelze přímo vložit ukazatel na privátní členskou metodu do funkce startování vlákna. Celý problém spočívá v tom, že vlákno musí být schopné funkci spustit samotnou bez instance třídy, která metodu vlastní. Aby bylo možné vykonávat metody SkenerUdalosti, musela jsem je převést na statické, čímž jsem přišla o ukazatel this na svoji instanci. Toto omezení se dalo obejít tím, že při startování vlákna je jedním z parametrů ukazatel typu void, který se předá do funkce vlákna při jeho spuštění. Ukázka č.6 ukazuje popsaný postup.

```
pthread_create(&(mVlakna[VLAKNO_GPS]),NULL, overitGPS, this);
pthread_create(&(mVlakna[VLAKNO_KLAVESNICE]),NULL, overitKlavesnici, this);

Udalost *udalost;
while(mBezi)
{
    udalost = 0;
    pthread_mutex_lock(&mMutext);
    if (!mFronta.empty())
    {
        udalost = mFronta.front();
        mFronta.pop();
    }
    pthread_mutex_unlock(&mMutext);
    // zbytek smycky chybi
}

//konec metody

void* SkenerUdalosti::overitGPS(void *skener)
{
    SkenerUdalosti *_this = (SkenerUdalosti *)skener;
    ...
```

Výpis 6: Ukázka tvorby vláken a použití mutexů

Vlákna běží, dokud má SkenerUdalosti spuštěnou smyčku. Jakmile se smyčka ukončí, čeká hlavní vlákno na vlákno pro GPS, které uzavře deskriptor a také se ukončí. Vlákno pro čtení z klávesnice se bohužel kvůli blokovanému režimu čtení kláves nedá šetrně ukončit a k jeho zničení dojde až při ukončení procesu aplikace.

SkenerUdalosti ve své smyčce prochází jednotlivé události ve frontě, pokud tam nějaké jsou. Každou událost zpracuje tím, že ji odešle naslouchajícím instancím observerové třídy Upozornitelni. Po odeslání události všem naslouchajícím objektům zkontroluje, zda-li není v události nastavena ukončovací akce, která ukončí hlavní smyčku, čímž se ukončí i celá aplikace. V tomto konkrétním případě má SkenerUdalosti zaregistrovaný pouze jeden observer, a to SpravceObrazovek, což popisuje schéma na obrázku č.4.



Obrázek 4: UML třídní diagram událostí

3.5.2 Události

Pro potřeby této aplikace bylo nutné vyvinout systém doručování událostí, a to proto, aby byla aplikace schopna individuálně řešit změny v GPS a stisky kláves. Jako báze slouží třída Udalost. Obsahuje enumerátory pro typ události TYP a pro druh akce po zpracování události POST. Pomocí typu události je potom možné událost přetypovat na konkrétní implementaci pro získání potřebných dat. Druh akce po zpracování je způsob, jak může příjemce události přikázat smyčce událostí, aby skončila, aniž by ji přímo znal. Konkrétní události budou popsány v navazujících podkapitolách a jsou zobrazeny v diagramu na obrázku č.4.

3.5.3 UdalostGps

Tato třída vychází z Udalost a jako typ vrací TYP_GPS. Tuto událost vytváří vlákno pro čtení dat z GPS přijímače a při jejím vytváření je nutné do konstruktoru předat ukazatel na konstantní strukturu nmeaINFO. GPS vlákno předává ukazatel přímo na vlastní strukturu, a tak nedochází nikde k jejímu kopírování. O životní zničení předaného pole se musí postarat ten, kdo UdálostGps vytvořil.

3.5.4 UdalostKlavesnice

Jedná se o velice jednoduchou událost, protože obsahuje pouze jeden byte informací, a to je stisklá klávesa. Jako svůj typ vrátí TYP_KLAVESNICE. Tuto událost vytvoří vlákno,

které čte z klávesnice pomocí ncurses. Není-li ncurses ve funkčním režimu, budou za každý stisk vygenerovány 3 události reprezentující celou escape sekvenci.

3.5.5 Správce obrazovek

Třída SpravceObrazovek obsahuje všechny obrazovky, které jsou v aplikaci. Jejím úkolem je distribuce událostí do aktivní obrazovky a také jejich přepínání. Přepnutí obrazovky je řešené odchycením události o stisku klávesy, kterou správce obrazovek nepošle dál. Pro přepínání slouží funkční klávesy a jejich výběr je specifikován jako konstantní symbol uvnitř třídy.

Obrazovky se nevytváří ve správci, ale je nutné je přidat pomocí funkce pridatObrazovku(). Nejdříve přidaná obrazovka bude zobrazena jako první, na ostatní je nutné se přepnout klávesnicí.

3.5.6 Obrazovka

Třída obrazovka slouží jako bázová třída pro všechny obrazovky. Tato třída umí

- připravit ncurses
- přijmout událost
- vytisknout titulek obrazovky
- vytisknout funkční menu
- vymezit prostor pro vytištění vlastního obsahu obrazovky

Konstruktor Obrazovky uloží její nadpis a zavolá metodu pripravCurses(), která nastaví knihovnu ncurses pro další použití. Metoda pripravCurses() se sice volá při konstrukci každého objektu, ale díky statické proměnné pripraveno k tomu ve skutečnosti dojde pouze jednou. Celá metoda je vidět ve výpisu kódu č.5.

Ve třídách, které dědí Obrazovku lze událost přijmout tím, že překryjí virtuální metodu naUdalost(). Jak dědící třída naloží s přijatou událostí je čistě na ní, ale jedná se o jediný prostředek, jakým se obrazovka dozví o změnách. Například třída ObrazovkaRadar v ní řeší navigaci pomocí kláves nahoru a dolů tím, že má statický iterátor, který inkrementuje nebo dekrementuje a předává ho do dalších tříd pro zvýraznění označeného řádku, což je ukázáno ve výpisu č.7.

```
void ObrazovkaRadar::naUdalost(const Udalost *udalost)
{
    switch(udalost->typ())
    {
    case Udalost::TYP_KLAVESNICE:
    {

        const UdalostKlavesnice *klavesnice = static_cast<const UdalostKlavesnice*>(udalost);
        static int i = 0;
```

```
        switch (klavesnice->klavesa()) {
        case KEY_DOWN: // navigace nahoru
            i++;
            if ( !mSatelity->vyberSatelit(i))
                i--;
            else
                mSilaSignalu->nastavAktualniRadek(i);
            break;
        case KEY_UP: // navigace dolu
            i--;
            if ( !mSatelity->vyberSatelit(i))
                i++;
            else
                mSilaSignalu->nastavAktualniRadek(i);
            break;
        case KEY_ENTER:
            break;
        }
        break;
    }
    // zbytek kodu je smazany, protoze neni dulezity pro ukazku
```

Výpis 7: Ukázka odchycení události v ObrazovkaRadar

Tisk obrazovky probíhá tak, že se nejdříve vytiskne nadpis obrazovky, poté se vyplní celá plocha obrazovky barvou pozadí a na spodním okraji obrazovky se vypíší funkční klávesy, které jsou přiřazeny jednotlivým obrazovkám, nakonec se zavolá virtuální metoda zobrazitObsah(), kterou implementuje každá obrazovka zvlášť. Existuje případ jako ObrazovkaObecna, kde je implementace zobrazitObsah(), ale nic se přímo v ní netiskne. Slouží totiž ke smazání stavu obsažené komponenty na zobrazování informací.

Aplikace obsahuje celkem 3 obrazovky. Každá z nich sdružuje určité komponenty a staví nad nimi základní logiku jako je navigace. První obrazovkou je třída ObrazovkaRadar, druhá je ObrazovkaDetaily a třetí je ObrazovkaObecne. ObrazovkaRadar obsahuje polohu všech viditelných satelitů a zároveň grafické znázornění jejich signálů. ObrazovkaDetaily opět znázornění poloh satelitů, ale k jednotlivým satelitům ukazuje přesnější informace. ObrazovkaObecna je souhrn veškerých informací, které je možné z GPS zobrazit.

3.5.7 Pomocné komponenty

Všechny obrazovky zobrazují konkrétní informace na základě událostí, ale neobsahují implementaci toho, jak tyto informace zobrazit, pouze předávají data dalším objektům, což jsou právě pomocné komponenty. Pomocné komponenty pomáhají udržet kód přehledný a zároveň umožňují svou znovupoužitelnost a vzájemnou komunikaci. Vzhledem k použití efektivní knihovny ncurses jsem se rozhodla naprogramovat reakce na změny dat také efektivně.

3.5.7.1 Satelity Třída Satelity slouží ke grafickému znázornění polohy satelitů a umožňuje výběr satelitu jeho podbarvením. Pracuje se s ní tak, že pro prvotní zobrazení stačí zavolat

metodu tisk(), která vytiskne obvodovou kružnici pro satelity. Ostatní tisknutí probíhá při předání dat o satelitech. Komponenta si pamatuje, jaká data naposledy zobrazila a při příchodu nových porovná, která se změnila, a podle toho aktualizuje obrazovku.

```
void Satelity :: satelity (const nmeaSATELLITE *satelity, int pocet)
{
    for(int i = 0; i < pocet ; i++)
    {
        if ( memcmp(&mPosledniSatelity[i], &satelity[i],(sizeof(int)*4)) != 0 )
        {
            prekresliOd( satelity ,pocet,i) ;
            break;
        }
    }
}
```

Výpis 8: Ukázka rozpoznání změn v komponentě Satelity

Ve výpisu č.8 je zobrazena implementace vyhledání změn. Jedná se o iterování skrz celý seznam satelitů, kde samotné porovnání provede funkce memcmp(), pro efektivnější použití jako velikost porovnávaných dat nepředávám sizeof(nmeaSATELITE), ale sizeof(int)*4, jelikož mě zajímají pouze první 4 hodnoty ze struktury nmeaSATELITE. Toto řešení by mohlo vést k problému při aktualizaci nmea knihovny, pokud by vývojáři změnili pořadí členů ve struktuře nebo přidali nového člena na začátek. Ovšem knihovna již nebyla delší dobu aktualizovaná, takže je tato možnost velmi nepravděpodobná.

Po nalezení změněné části se zavolá metoda prekresliOd(), která zajistí vymazání změněných satelitů pomocí seznamu, jenž si tvoří při tisknutí satelitů. Dále sesynchronizuje vlastní seznam satelitů a vytiskne satelity od změněného satelitu. Tento postup jsem si zvolila, protože není tak riskantní a komplikovaný jako řešení přesunutých satelitů podle jejich ID nebo odstranění ze seznamu a zajistí mi bezpečný provoz.

Komponenta provádí veškerý tisk, včetně obvodové kružnice, přes metodu tiskBodu(). Pro názornost bude lepší vypsat konkrétní implementaci ve výpisu č.9. Nejdříve bych vysvětlila problém, který jsem měla při prvotní implementaci. Při prvním vykreslování kružnice byla moje kružnice šišatá, eliptická. Způsobil to fakt, že body kterými tisknu kružnici nejsou ve skutečnosti body ale obdélníky, což mě donutilo pro tisk kružnice použít vzorec pro elipsu, která bude kopírovat poměr stran mého obdélníkového bodu. Snadno jsem zjistila, že písmo na mém počítači je v poměru 8.0/15.0, což je konstanta, kterou přenásobuji poloměr a výsledek ukládám do členské proměnné mUpravenaVyska. Metoda tiskBodu přijímá jako parametry úhel, ve kterém je bod na kružnici, řetězec pro tisk, poloměr kružnice a informaci, zda-li se má vytištěný text podbarvit.

Metoda se chová dvěma způsoby podle toho, jestli se specifikuje poloměr nebo ne. Poloměr a podbarvení totiž nejsou povinné parametry. Není-li při volaní zadán poloměr, funkce automaticky počítá s tím, že tiskne obvodový kruh a použije předem vypočítané hodnoty. V opačném případě dojde k přepočtu zadaného poloměru v poměru stran. Následně se vytiskne bod a jeho vypočítané souřadnice se z metody vrátí. Tisk obvodové kružnice se provádí cyklováním úhlu po jednom stupni.

```
Bod Satelity :: tiskBodu(int uhel,const char* znak, double polomer,bool podbarvit)
```

```
{
    double spolomer;
    double vpolomer;
    if (polomer < 0)
    {
        spolomer=mPolomer;
        vpolomer=mUpravenaVyska;
    }
    else
    {
        spolomer=polomer;
        vpolomer=(mUpravenaVyska/(double)mPolomer)*polomer;
    }
    double rad = ((2.0 * M_PI) / 360.0) * (double)uhel;
    Bod bod;
    bod.x = mStredX + (cos( rad ) * spolomer);
    bod.y = mStredY - (sin( rad ) * vpolomer);
    if (podbarvit)
        wattrset(stdscr, COLOR_PAIR(E_PODBARVENI));
    else
        wattrset(stdscr, COLOR_PAIR(E_BARVA_POZADI));
    mvwprintw(stdscr, bod.y ,bod. x,"%s",znak);
    return bod;
}
```

Výpis 9: Ukázka tisku bodu v kružnici

Nepostradatelná je také metoda reset(), která nastaví komponentu do výchozího stavu. Výchozím stavem se myslí prázdný seznam uložených satelitů a vynulování člena specifikujícího, který satelit se má podbarvit.

Třída Satelity slouží také pro realizaci navigace mezi satelity, k čemuž využívá metodu vyberSatelit(). Tato metoda se pokusí nastavit podbarvení předaného satelitu a při úspěchu vrací true. Toho využívají třídy ObrazovkaDetaily a ObrazovkaRadar, které při navýšení jejich interního iterátoru předaného do vyberSatelit(), testují jeho úspěšnost a podle toho buď vrátí iteraci zpět nebo předají ověřený iterátor jiným komponentám.

3.5.7.2 Síla signálu Třída SilaSignalu je indikátor síly signálů všech dostupných satelitů. Této komponentě se při vytváření předávají její počáteční souřadnice a šířka. Jedná se o seznam, který graficky i textově znázorňuje sílu signálu. Položky v seznamu je možné podbarvovat, a tím docílit navigace v seznamu. Stejně jako předchozí komponenta má i tato metodu reset(), pro nastavení výchozího stavu, a třída si pamatuje stavy vykreslených satelitů, podle nichž efektivně překresluje jen ty změněné. V případě této komponenty bylo zajímavé vyřešit problém tisknutí textu síly signálu a jeho podbarvení, protože se často stávalo, že grafické znázornění síly signálu zasahovalo do jeho textu.

Vykreslování se muselo rozdělit na část, která zjistí dopředu výsledný textový řetězec a následně algoritmus kreslení grafické části rozseknout na dvě části. První část tiskne grafické znázornění a zároveň při tom používá písmena z již vytvořeného řetězce, dokud jsou. Druhá část ověří, zda-li nezbyly nějaké nevytištěné znaky v oblasti, kde už grafické znázornění nepokračuje.

Oproti předchozí komponentě se tato komponenta nijak netiskne předem, veškerý výstup se děje až při předání konkrétních dat k zobrazení.

3.5.7.3 GPS Info Třída GpsInfo vypisuje všechna dostupná data o pozici a satelitech. Při konstrukci je třeba předat pozici levého horního rohu. Aktuálně má z knihovny nmea k dispozici informace

- o čase
- fix
- o použitých satelitech
- o viditelných satelitech
- o zeměpisné šířce
- o zeměpisné výšce
- výšce nad mořem
- rychlosti
- o směru pohybu
- odchylce pozice
- horizontální odchylce
- vertikální odchylce

Tato komponenta čerpá převážně z dat ze struktury nmeaINFO popsané v tabulce č.2, ve které je ukázáno, že zeměpisná šířka i délka je udávaná ve formátu NDEG - +/- [stupně][min].[sec/60], pro který jsem napsala parsovací metodu z ukázky č.10.

```
void GpsInfo::naStupne(double vstup, int& stupne, int& minuty, double& vteriny)
{
    stupne = ((int)vstup)/100;
    minuty = (int)vstup − stupne ∗ 100;
     vteriny  = (vstup − double(stupne∗100 + minuty))∗60.0;
}
```

Výpis 10: Parsování polohy z NDEG formátu

Jako první komponenta nemá podporu pro jakýkoliv jiný vstup než nmeaINFO, protože by u ní pokus o jakoukoliv navigaci postrádal smysl. Metoda reset() pro nastavení výchozích hodnot zde dostupná je.

3.5.7.4 Satelit info Třída SatelitInfo slouží k vypsání všech dostupných informací o jednom konkrétním satelitu. Komponentě se při vytváření zadává pozice levého horního rohu. I zde je aktualizace dat na obrazovce změněna pouze v případě, že se novými daty změnil zobrazený satelit. Zavoláním metody vypsatSatelit() s parametrem pořadí, dojde k vypsání informací o požadovaném satelitu, pokud jsou jeho data již předána metodou satelity().

4 Kompilace a spuštění

Vše o zdrojovém kódu je již popsané, nyní zbývá aplikaci zkompilovat a spustit.

4.1 Kompilace

Pro úspěšnou kompilaci je nutné mít nainstalované hlavičkové soubory knihovny ncurses. Knihovna nmea je s projektem přiložena a linkuje se staticky, takže po kompilaci již nebude třeba. Nejprve je nutné si zkompilovat libnmea, která má vlastní Makefile a to spuštěním příkazu make ve složce lib/nmealib/. Následně je možné spustit make v kořenovém adresáři projektu. Výstupem projektu je spustitelný binární soubor gps-reader.

4.2 Spuštění

Pro spuštění je nutné mít nainstalovanou knihovnu ncurses a mít připojený GPS modul. Velice důležité je, aby měla aplikace práva pro čtení z GPS příjmače. Toto lze nastavit jednorázově pomocí programu chmod[2] nebo trvale vytvořením pravidla, např. v systému udev. Aplikaci nijak neovlivní, v jaké pracovní cestě bude spuštěna, jelikož jediný výstup, který generuje, je konzolový a dále zpracovává vstup z GPS modulu, který sama detekuje.

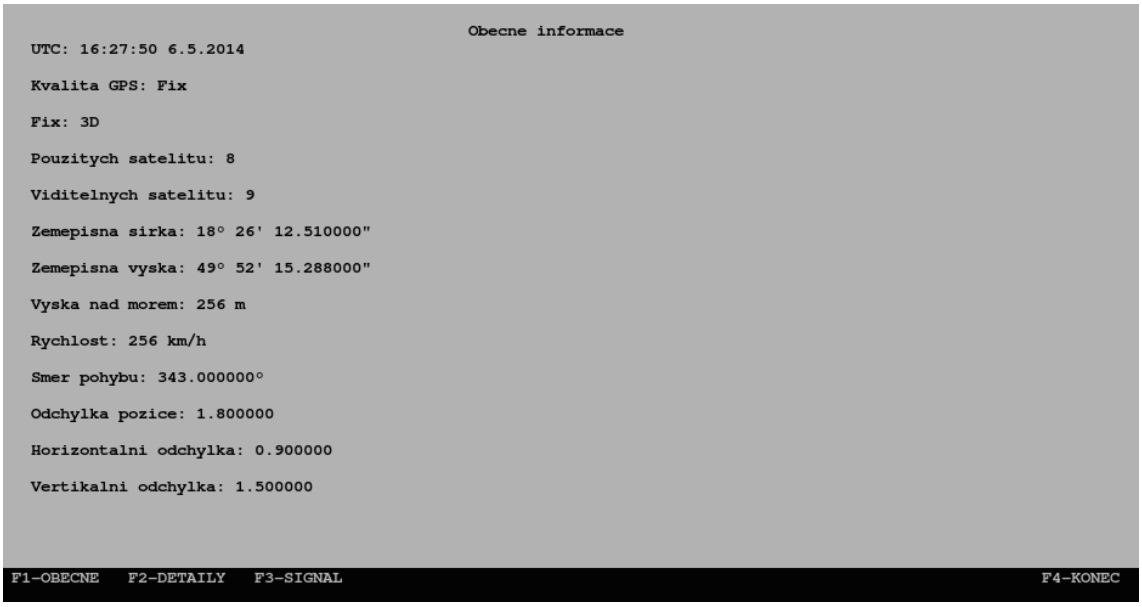
Při spuštění aplikace dojde k detekci GPS modulu a následně otevření první obrazovky. Pokud se modul nedetekuje, je nutné ho zadat jako parametr při spouštění ve formátu *./gps-reader <cesta k zařízení>*, což by mohlo být např. *./gps-reader /dev/ttyACM0*. Může se také stát, že aplikace nalezne více zařízení, odpovídajících kritériím detekce, v tom případě je vypsán očíslovaný seznam zařízení, ze kterých si lze vybrat.

Pokud je vše v pořádku, měly by se objevit obecné informace o GPS, jako na obrázku č.5. Zde se průběžně aktualizují obecné informace a jediná možná navigace je přepnutí se na další obrazovku. Stiskem klávesy F2 dojde k přepnutí na obrazovku s grafickým znázorněním polohy satelitů a detailů o vybraném satelitu. Na této obrazovce je možné klávesami nahoru a dolů přeskakovat v grafickém znázornění satelitů po jednotlivých satelitech. Výběr lze rozpoznat červeným písmem, přičemž u takto označeného satelitu se vypíše detail v pravé polovině obrazovky, což je viditelné na obrázku č.6.

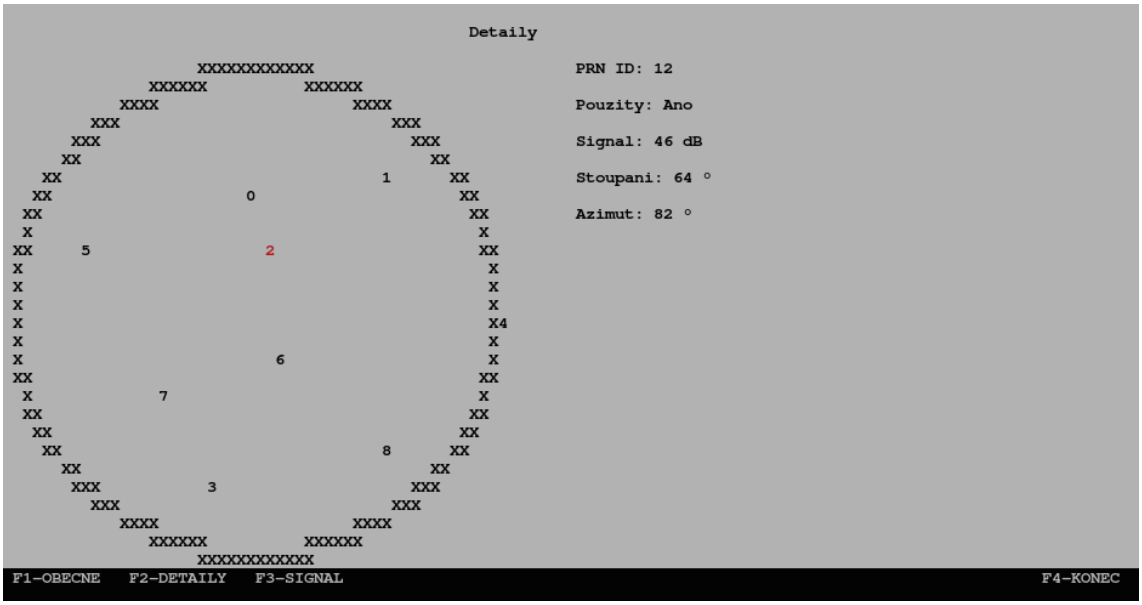
Na poslední obrazovku je možné se přesunout stiskem klávesy F3. Na této obrazovce je dobře vidět síla signálu na jednotlivých satelitech, jelikož je zobrazena graficky spolu s grafickým znázorněním polohy satelitu, jako na předchozí obrazovce. Opět je možné se šipkami nahoru a dolů přepínat mezi satelity, zde je zvýrazněno červenou barvou číslo vybraného satelitu na obou zobrazených komponentách. Příklad, jak by obrazovka mohla vypadat, je na obrázku č.7.

Na závěr bych podotkla, že velikost terminálu ovlivní kvalitu grafického znázornění družic a to tak, že čím větší bude počet řádků a sloupců v terminálu, tím bude výstup přesnější a přirozenější. Dále mezi každým přepnutím obrazovek dojde ke krátké prodlevě v zobrazení nových dat, což je způsobeno tím, že obrazovka vypíše data až po příchodu události s daty GPS. Aplikaci je možné ukončit stiskem klávesy F4.

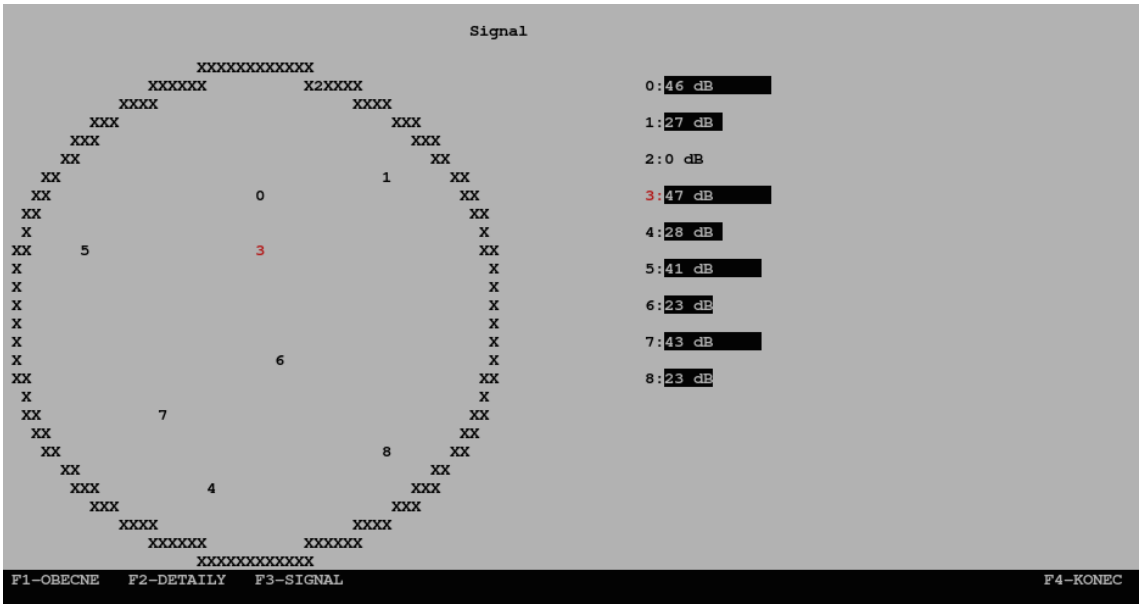
[2] Detaily v manuálu příkazem *man 1 chmod*



Obrázek 5: Obrazovka obecných informací o GPS



Obrázek 6: Obrazovka s detaily vybraného satelitu, výběr je vyznačen červeně



Obrázek 7: Obrazovka se sílou signálu satelitů, výběr je vyznačen červeně

5 Závěr

Mým úkolem bylo naprogramovat nativní Linuxovou aplikaci, která se bude schopna spojit s GPS přijímačem pomocí protokolu NMEA a zpracovat informace o satelitech do grafické formy. Jelikož moje aplikace obsahuje vše, co bylo v zadání, usuzuji, že jsem byla při vypracování práce úspěšná. Ovšem nemůžu říct, že bych nyní určité části aplikace nenapsala jinak. Myslím si, že volba lepšího rozhraní pro třídy komponent, by byla na místě a zároveň by návrh nového rozhraní mohl více využívat knihovnu ncurses. Například řešení vestavěné funkce pro posouvání/rotaci seznamu by bylo s aktuálním návrhem velice obtížné. Komponenta ani obrazovka nemají nástroj, kterým by se mohly spojit s jinou komponentou či obrazovkou, což v rozsahu řešení, které jsem připravila, úplně nevadí, ale bylo by vhodnější, kdyby např. obrazovka byla schopna zaslat požadavek o přepnutí na jinou obrazovku, aktuálně je tento proces pevně zakódovaný ve třídě SpravceObrazovek.

Jelikož aplikace poskytuje pouze jednoduché rozhraní, umožňující číst přímá data z GPS přijímače, umím si představit, že by se na tomto programu dalo stavět něco užitečnějšího, např. zasílání událostí GPS na D-Bus podle specifikovaných filtrů, což by umožnilo dalším aplikacím snadný přístup k GPS datům. Samotná konzolová aplikace by se dala rozšířit o komponentu jako je kompas, který by směřoval do bodu zadaného souřadnicemi. Bylo by zajímavé rozšířit aplikaci o podporu čtení z více než jednoho přijímače, pomocí které by se daly dělat dodatečné korekce vzájemných vstupních dat.

6 Reference

[1] Matthew, Neil & Stones, Richard *Linux Začínáme programovat*, Brno: Computer Press, a. s., 2008.

[2] Čábelka, Miroslav *Úvod do GPS, Skriptum Přírodovědecké fakulty*, Praha: UK v Praze, Přírodovědecká fakulta, 2008.

[3] Šebesta, Jiří *Globální navigační systémy, Skriptum FEKT*, Brno: VUT v Brně , FEKT, 2012.

[4] Český kosmický portál. Informační stránky Koordinační rady ministra dopravy pro kosmické aktivity [online]. URL: ¡`http://www.czechspaceportal.cz/`¿ [vid. 2014-4-1]